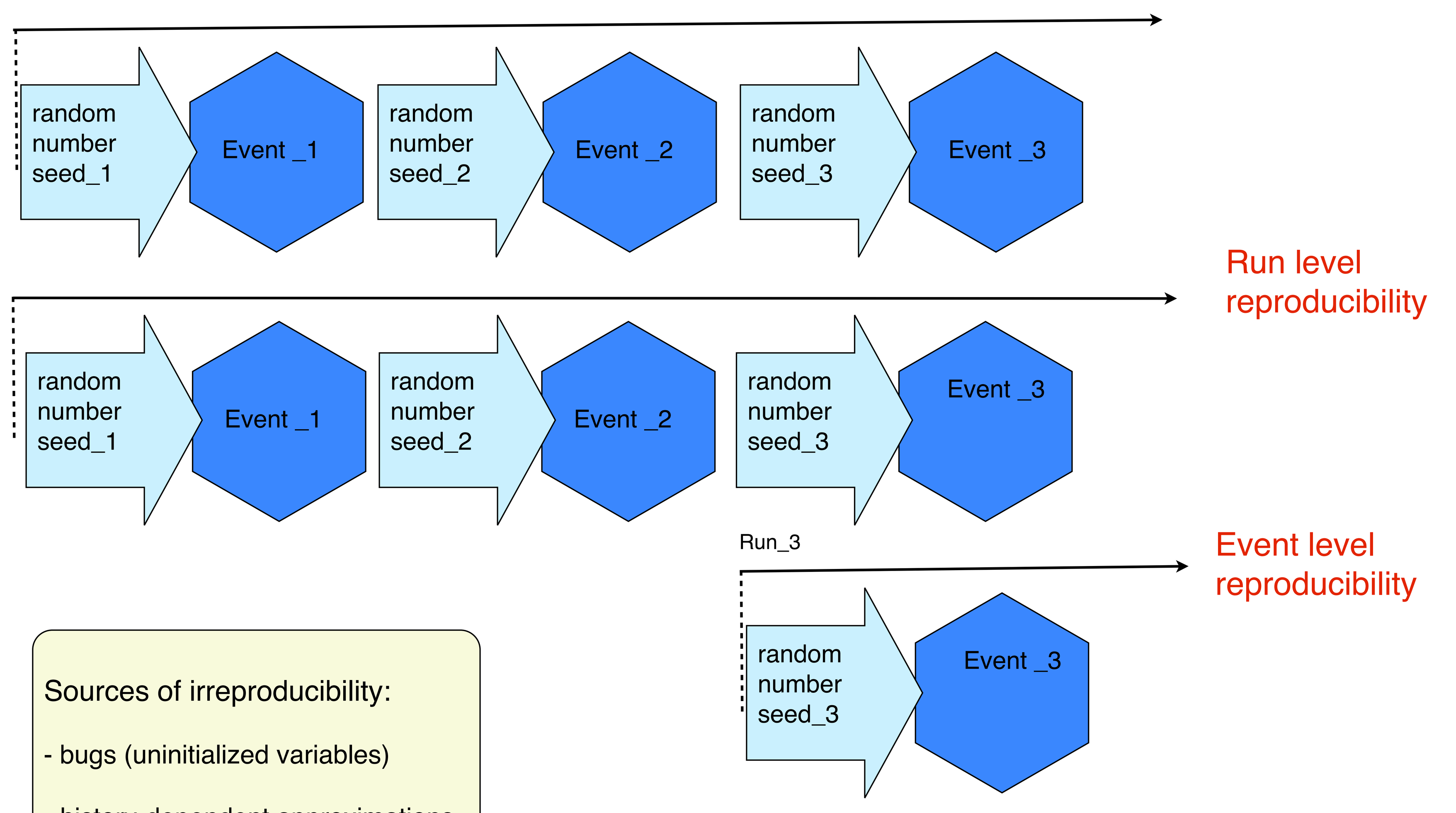
# Recent Developments in the Geant4 Hadronic Framework

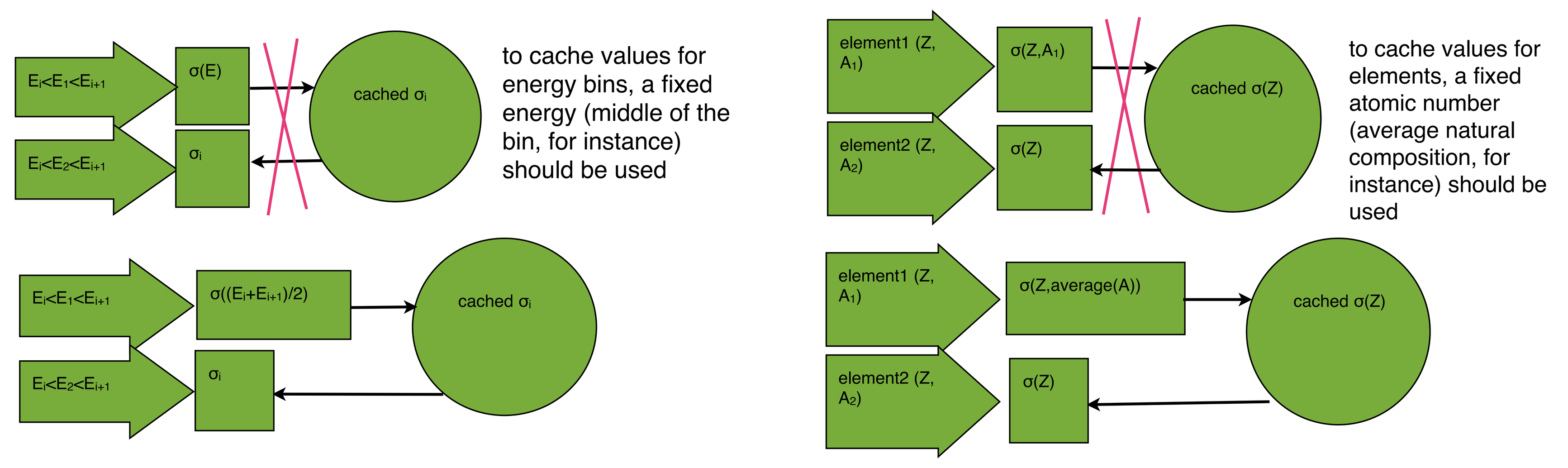
Witold Pokorski (CERN), Alberto Ribon (CERN)

**Geant4** is the main simulation toolkit used by the **LHC** experiments and therefore a lot of effort is put into improving the physics models in order for them to have more predictive power. As a consequence, the code complexity increases, which requires constant improvements and optimizations on the programming side. In this poster, we discuss the recent developments and improvements in the **hadronic framework** of the Geant4 simulation toolkit.

## Fixing reproducibility of simulated events in hadronic physics code

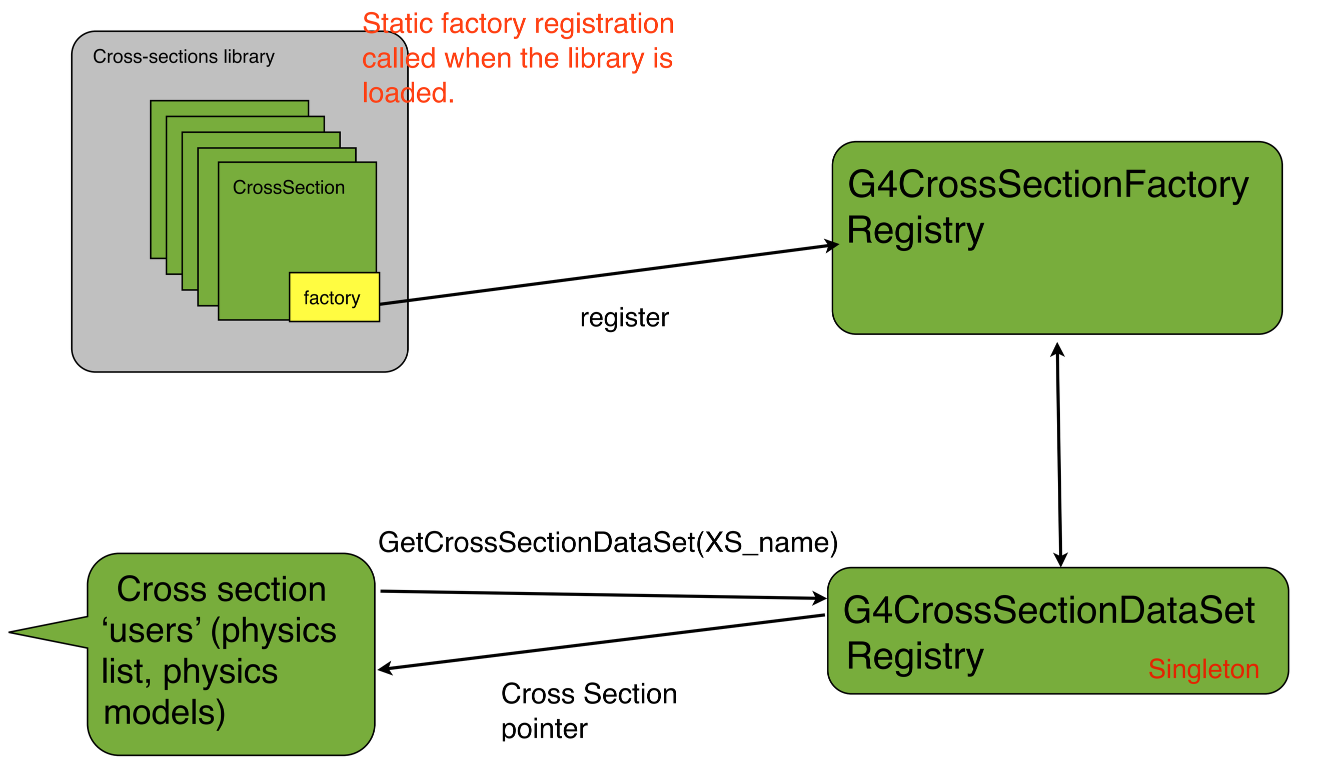
- use of pseudo-random number generator implies that events should be reproducible
  - non-reproducibility of events makes it difficult to debug the code
- run level reproducibility (starting from the same random-number generator seed)
- event level reproducibility (starting from any event within a run).




## Sharing hadronic cross-sections via factory pattern

- goal is to share cross-section objects between different 'users' (physics processes, models, physics lists, etc)
- factory pattern introduced for the instantiation of the cross section objects
- extended functionality of G4CrossSectionDataSetRegistry to store and to provide the pointer to cross section objects.




➡ 'Cross-section user' asks G4CrossSectionsDataSetRegistry for a given G4CrossSectionDataSet by specifying its name (string).

➡ The registry checks if this cross-section has been already instantiated.

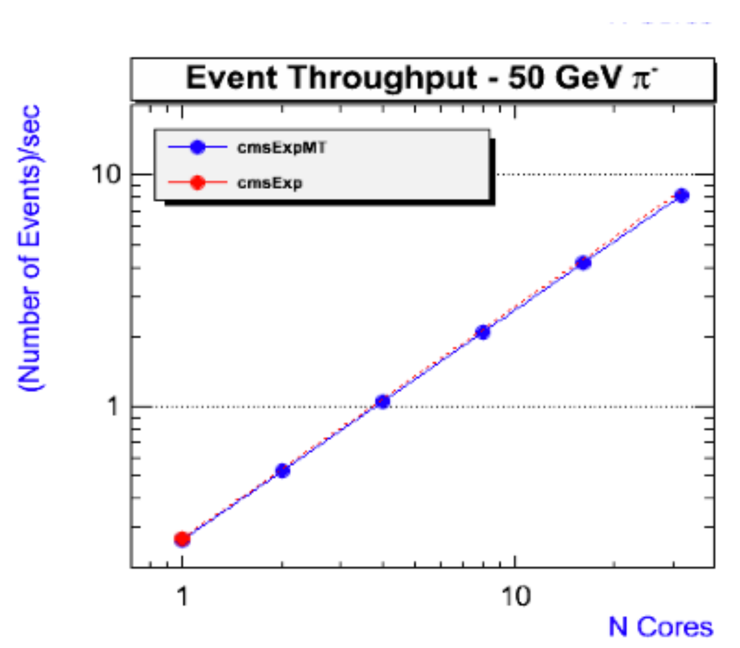
➡ If yes, it returns the pointer to it (shared between all 'cross-section users').

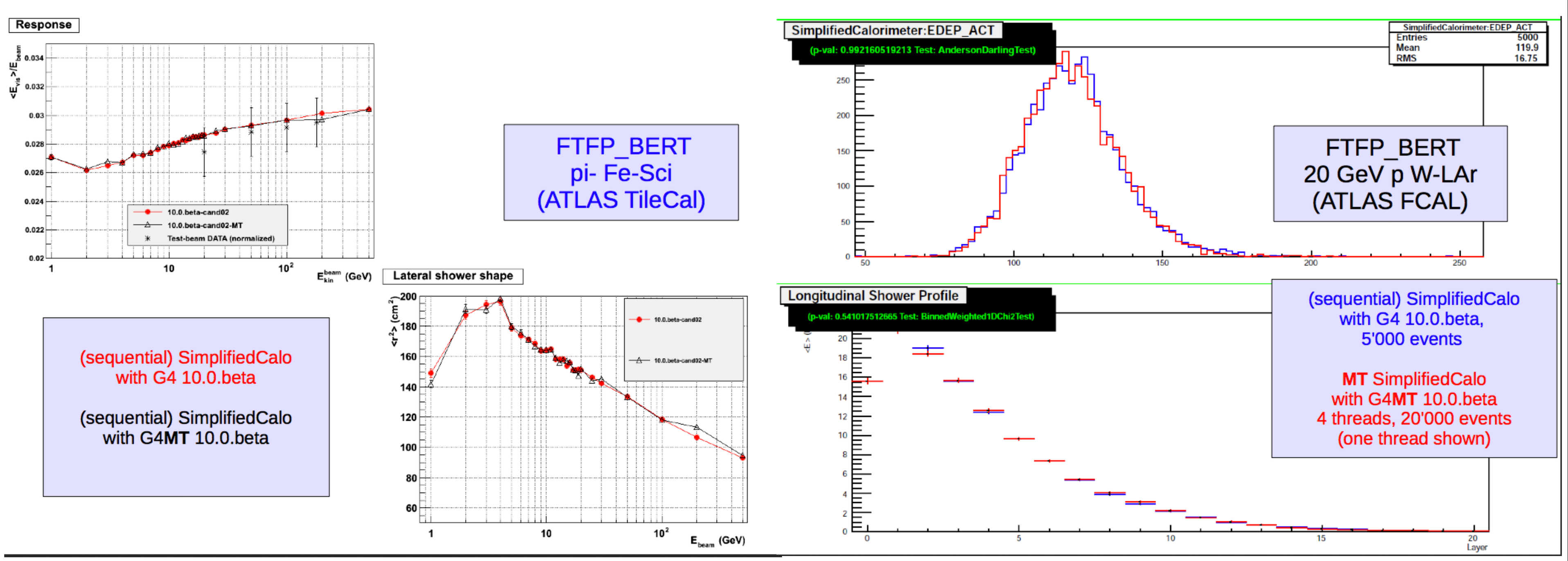
➡ If not, the registry uses the factory to instantiate the given cross-section. If the factory does not exist, it return an error 'cross-section not found'.

## Multi-Threading in Geant4 hadronic physics

- number of technical issues addressed to eliminate any interference between several threads accessing the hadronic physics classes
- objects that can be easily shared are those that are read only
- caching becomes tricky because of possible simultaneous write access to cache
- in order to validate the multi-threaded code we require that the calorimeter (and other) observables remain statically the same between sequential and multi-threaded modes.

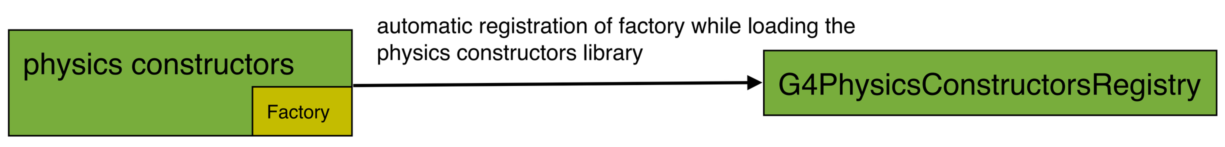
- multi-threading increases significantly the event throughput
- challenge is the reproducibility of events
  - a number of fixes and improvements to achieve full reproducibility




## Run-time configuration of hadronic physics list

➡ Generic Physics list class removes the compile- and link-time dependency between the users code and the specific physics models
➡ introduced registry of physics "constructors"
➡ instrumented physics "constructors" to provide factories that get registered in the registry

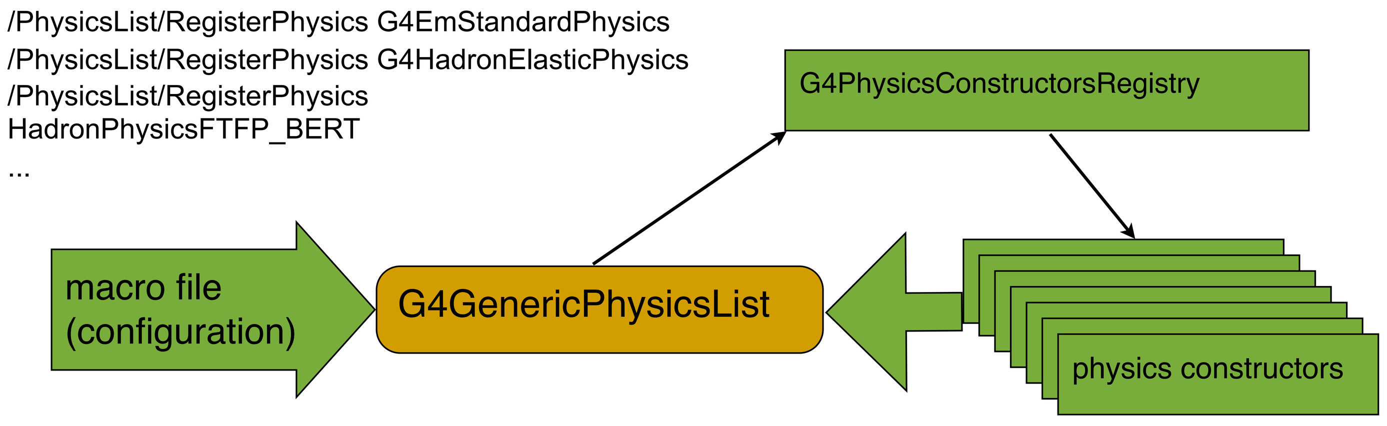



Physics Lists can now be constructed in two new ways:

➡ through G4GenericPhysicsList class using a macro file:

```
FTFP_BERT.mac:
/PhysicsList/defaultCutValue  0.7
/PhysicsList/SetVerboseLevel 1

/PhysicsList/RegisterPhysics G4EmStandardPhysics
/PhysicsList/RegisterPhysics G4HadronElasticPhysics
/PhysicsList/RegisterPhysics
HadronPhysicsFTFP_BERT
...
```

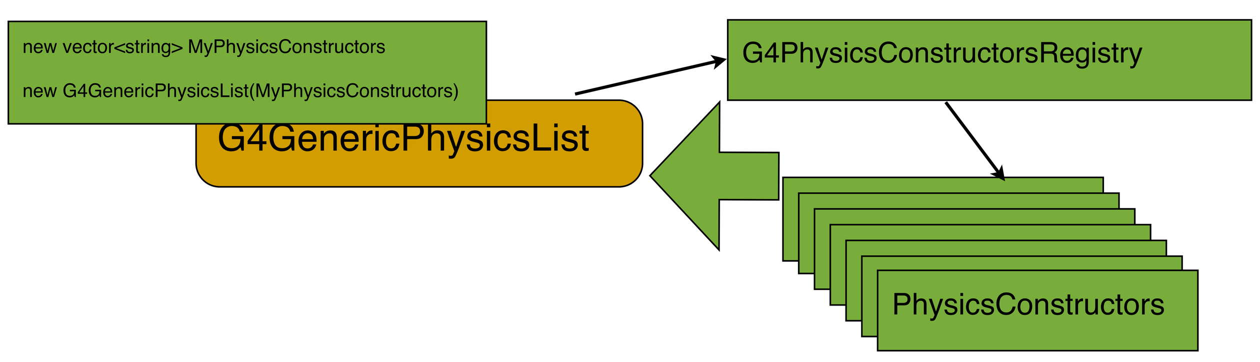



and the main() containing:

```
phys = new GenericPhysicsList();
G4UImanager* UImanager = G4UImanager::GetUIpointer();

UImanager->ApplyCommand("/control/execute FTFP_BERT.mac");
```

➡ by passing a vector of physics 'constructors' names at the instantiation time

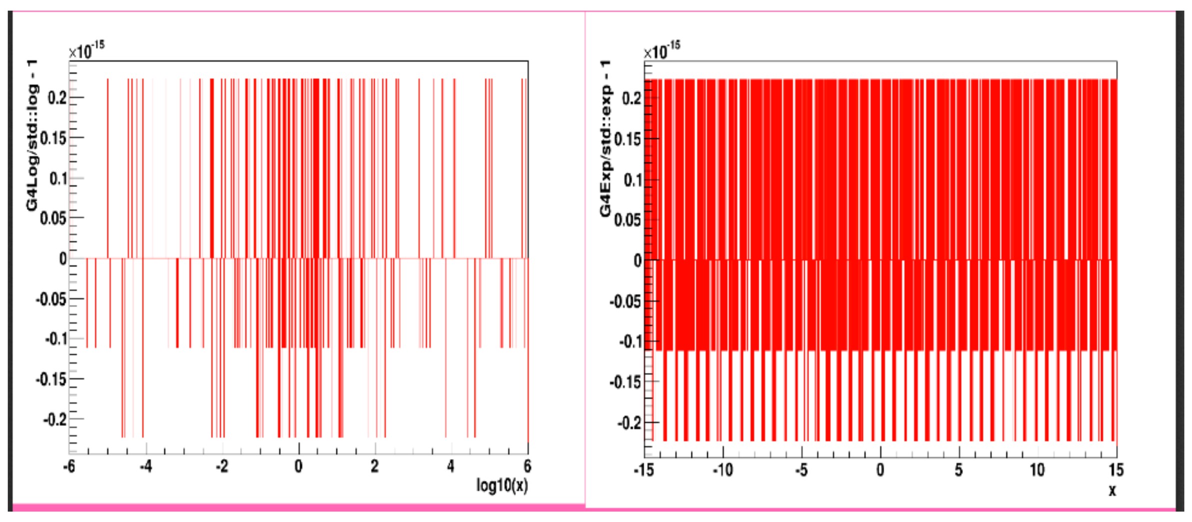



## Use of fast mathematical functions

- precision of hadronic cross sections is at the level of 5-10%
  - no need to do there high-precision calculations
- using fast log and exp functions increases CPU performance
  - no significant lost in the precision of the simulation results
- replaced the std::log and std::exp by faster implementation from VDT library (see Danilo Piparo talk at CHEP 2013).
  - effect is much below the precision of the cross-sections,
- the calculation of the cross-sections values was faster by ~5%.

| | Std | G4 VDT | G4Pow |
|---|---|---|---|
| Log | 8.97 | 4.91 | 5.19 |
| Exp | 13.93 | 1.95 | 1.34 |
| $A^{1/3}$ | 20.46 | 7.03 | 0.77 |
| $Z^{1/3}$ | - | - | 0.01 |

Comparison of CPU performance of different implementations of mathematical functions




Precision of VDT functions